ComPAS'2014 : Parallélisme/ Architecture / Système
Neuchâtel, Suisse, du 23 au 25 avril 2014

Contrôle d'application flot de données pour les systèmes sur puces : étude de cas sur la plateforme Magali

Mickaël Dardaillon *[1], Kevin Marquet[1], Tanguy Risset[1], Jérôme Martin[2], Henri-Pierre Charles[3]

[1]Université de Lyon, Inria,
INSA-Lyon, CITI-Inria, F-69621, Villeurbanne, France
prenom.nom@insa-lyon.fr

[2] Univ. Grenoble Alpes, F38000 Grenoble, France
CEA, LETI, Minatec campus, F-38054 Grenoble, France
jerome.martin@cea.fr

[2] Univ. Grenoble Alpes, F38000 Grenoble, France
CEA, LIST, Minatec campus, F-38054 Grenoble, France
henri-pierre.charles@cea.fr

Résumé
Les applications embarquées demandent toujours plus de puissance de calcul pour moins de consommation, avec comme conséquence l'apparition de systèmes sur puces dédiés. Dans le domaine du traitement du signal, le modèle de calcul flot de données est couramment utilisé pour la programmation de ces systèmes sur puce. Il est donc nécessaire d'avoir un modèle d'exécution adapté à ces architectures et répondant aux contraintes applicatives. Dans ce travail, nous proposons un nouveau modèle d'exécution pour le contrôle d'applications flot de données. Notre approche s'appuie sur les liens entre les caractéristiques des applications et les performances selon le modèle d'exécution associé. Ce travail est illustré avec une étude de cas sur la plateforme Magali.

Mots-clés : flot de données, contrôle centralisé, calcul distribué

---

1. Introduction

Les algorithmes de traitement du signal requièrent une grande puissance de calcul, dépassant les limites du budget énergétique d'ordinateurs généralistes. La programmation flot de données est un champ de recherche actif et de nombreux environnements populaires (Simulink, LabView, etc.) utilisent ce paradigme pour le design et l'implémentation d'algorithmes de traitement du signal. En effet, les caractéristiques de ce modèle de calcul (échange de données explicite, parallélisme de données et de tâches, etc.) permettent d'atteindre de grandes performances de calcul et d'efficacité énergétique, le rendant adapté aux applications embarquées tels que la cryptographie, les télécommunications, le décodage audio et vidéo, etc.
Le champ des télécommunications numériques sans fil fournit régulièrement de nouveaux challenges technologiques ; on peut mentionner l'arrivée des protocoles de communication dit

∗. Cette thèse est financée par une Allocation Doctorale de Recherche de la Région Rhône Alpes 11 01302401.

de quatrième génération (LTE-Advanced), ou le développement des réseaux radio cognitifs. Ces tendances technologiques ont réactivé la recherche dans la compilation d'application flot de données dynamique ou les algorithmes de traitement de données distribués. En particulier, le besoin de programmes flexibles mais vérifiables a conduit à l'apparition de modèles de calcul flot de données paramétriques.
Un exemple type de la dynamicité dans les applications flots de données vient du protocole LTE-Advanced : le type de modulation (par exemple QPSK, 16QAM) utilisé dans une trame est contenu dans la trame elle-même. Le système doit donc être capable de s'adapter à cette modulation en quelques microsecondes.
Le décodage du LTE-Advanced, qui doit être embarqué dans la prochaine génération de téléphones portables, requière des améliorations radicales de performances. Ces systèmes incluront des puces dédiées avec suffisamment de puissance de calcul (de l'ordre de 40 GOPS [25, 7]) pour une consommation raisonnable (moins de 500 mW). Cela nécessite l'utilisation d'architectures spécifiques, possédant des blocs d'accélération matérielle dédiés, hétérogènes et distribués, avec des mécanismes de synchronisation adaptés. Ces nouvelles plateformes matérielles programmables se concentrant sur le domaine des télécommunications sont connues sous le nom de *radio logicielle*.
La problématique posée est l'adaptation du modèle d'exécution aux contraintes applicatives d'une part, et à la plateforme matérielle d'autre part. Le choix du modèle d'exécution associé au modèle de calcul est en effet primordial, puisqu'il est en charge du contrôle de l'ensemble de l'architecture et de la synchronisation entres ses différents éléments, et doit répondre aux contraintes posées par notre cadre applicatif.
Dans la suite de l'article, nous décrivons tout d'abord les différentes architectures matérielles répondant à nos contraintes, ainsi que les chaînes de développement permettant de les programmer en section 2. L'environnement matériel et logiciel autour de Magali ainsi qu'un nouveau modèle de calcul sont introduits en section 3. La section 4 présente l'influence des caractéristiques d'application sur les modèles d'exécution, illustré par des résultats d'implémentation.

2. État de l'art

Les contraintes applicatives dans le domaine des télécommunications donne lieu au développement de nouvelles plateformes matérielles hétérogènes. Ce développement s'accompagne de nouveaux environnements de programmation permettant d'adresser les spécificités de ces plateformes. Afin de mieux comprendre ces spécificités et leur influence sur ces nouveaux environnements de développement, nous allons dans un premier temps passer en revue les plateformes matérielles de radio logicielle, avant de nous pencher sur leur environnements de développement.

2.1. Architectures matérielles pour la radio logicielle

Parmi les radios logicielles, un certain nombre utilisent une architecture de type accélérateur, dans lequel un processeur principal est associé à un processeur dédié chargé d'accélérer les parties lourdes en calcul. Nous présentons ici quelques exemples représentatifs.
La plateforme BEAR [20] développée par l'Imec est une architecture reconfigurable à gros grain, construite autour d'un processeur central et de l'accélérateur ADRES [5]. L'ADRES est vu par le processeur principal comme un processeur VLIW, composé d'une grille de 16 unités fonctionnelles. Chacune de ces unités est un processeur SIMD, tirant parti du parallélisme de données. Cette approche est similaire à celle consistant à utiliser le processeur graphique présent dans un ordinateur généraliste afin d'accélérer une partie des calculs (GPGPU : *General-purpose Proces-*

sing on Graphics Processing Units). Des expérimentations dans ce sens ont d'ailleurs été réalisées en radio logicielle [15].
Ces architectures offrent un degré limité de parallélisme. Afin de répondre à ce besoin, de nouveaux systèmes sur puce dédiés ont été développés. Ils se composent d'un ensemble de processeurs et d'accélérateurs matériels, généralement accompagné d'un processeur généraliste servant de contrôleur central. Nous présentons quelques architectures significatives de cette approche.
Infineon a construit le circuit MuSIC [21] comme solution pour la radio logicielle. Un processeur ARM est en charge du contrôle centralisé. Le traitement du signal est assuré par 4 DSPs SIMD et plusieurs accélérateurs dédiés pour le filtrage et le codage canal. Les architectures Sandblaster [23] et ARDBEG [26] sont construites de manière similaire, autour d'un contrôleur ARM, de 4 DSPs ainsi que des accélérateurs dédiés.
La plateforme Magali [8] est elle construite autour d'un réseau sur puce (Network-on-Chip), chaque périphérique ayant accès au réseau, avec un contrôleur centralisé ARM. On retrouve des DSPs pour les parties calculatoires, plusieurs blocs d'accélération matérielle dédiés aux applications télécoms, ainsi que de la mémoire distribuée associée à des DMA. L'ensemble des blocs d'accélération matérielle est doté de mécanismes d'accélération matérielle pour le contrôle distribué. On peut citer Tomahawk [17] comme architecture similaire, ainsi que Genepy [16] comme évolution de Magali.
Dans cet état de l'art, nous avons principalement rencontré deux tendances du point de vue contrôle : 1) les plateformes de type accélérateur se rapprochant du GPGPU et 2) les plateformes distribuées possédant un contrôleur centralisé ainsi que des mécanismes de contrôle distribués. Nous allons maintenant voir les environnements permettant de programmer chacune de ces tendances.

2.2. Environnement de développement pour la radio logicielle

Pour un programmeur de plateforme embarquée, une manière simple de développer sur une plateforme de radio logicielle est d'utiliser un langage impératif associé à des fils d'exécution (*threads*) afin d'exprimer le parallélisme. Sur les architectures de type accélérateur, la similitude au GPGPU permet d'envisager la réutilisation des travaux existants dans ce domaine. L'environnement EXOCHI [24] est un exemple proposant d'étendre OPENMP avec des fonctions intrinsèques placées dynamiquement sur un système hétérogène en fonction des ressources disponibles. On peut aussi mentionner l'environnement associé à la plateforme BEAR [20] permettant de la programmer à partir d'une application Matlab.
Des travaux récents poussent à un changement de paradigme permettant une meilleure abstraction de ces plateformes à l'aide de langages flot de données. Ces langages reposent sur l'utilisation d'un modèle de calcul où le programme est représenté comme un graphe dirigé $G = (V, E)$. Un acteur $v \in V$ représente un module de calcul ou un sous graphe hiérarchique. Un arc dirigé $e \in E$ représente une mémoire tampon depuis l'acteur source S jusqu'à sa destination D. Les graphes flots de données suivent une exécution dirigée par les données : un acteur v peut être exécuté uniquement quand suffisamment de données sont disponibles sur ces arcs d'entrées. Quand il est exécuté, v consomme un certain nombre d'éléments sur ces arcs d'entrées et en produit un certain nombre sur ces sorties. Le dynamisme exprimable varie en fonction du modèle de calcul flot de données utilisé, tous n'étant pas adaptés à l'expression d'applications de radio logicielle [3].
ΣC [14] est un langage de programmation utilisant une extension de C. Le modèle de calcul associé permet d'exprimer des applications dynamiques tout en autorisant un ordonnancement statique. Cet environnement vise les SoCs multi-coeurs. MAPS [6] est aussi basé sur une exten-

sion de C, et utilise le modèle de calcul flot de données dynamique. Ce dynamisme donne lieu à un ordonnancement et placement dynamique, avec des opérateurs de traitement du signal placés sur les accélérateurs dynamiquement lors de l'exécution.
L'environnement SPEX [18] propose de programmer à l'aide de trois paradigmes. La programmation séquentielle appelée *Kernel* SPEX permet de spécifier les traitements de données. Un paradigme flot de données dynamique est utilisé dans *Stream* SPEX pour lier ces traitements. Le langage synchrone *Synchronous* SPEX représente la couche de contrôle.
Les tendances matérielles se reflètent dans la manière de programmer ces plateformes. On retrouve pour les architectures de type accélérateur des approches impératives enrichi d'abstraction permettant d'exploiter les ressources hétérogènes. Pour les plateformes distribuées, la tendance de nombreux environnements est l'utilisation du paradigme flot de données pour leur programmation. Nous allons concentrer la suite de notre étude sur ce paradigme. À cet effet nous utiliserons la plateforme Magali, son architecture distribuée et hétérogène permettant de représenter les contraintes matérielles des plateformes visées.

3. Plateforme Magali

Nous présentons maintenant la plateforme Magali afin de pouvoir expliciter les contraintes de notre cas d'étude. Pour cela, nous allons commencer par décrire la plateforme matérielle Magali en nous attardant sur les mécanismes de contrôle proposés. Nous étudierons ensuite les différentes manières existantes de programmer cette plateforme et en particulier les mécanismes de contrôle, avant de décrire celle utilisée par notre chaîne de compilation à l'origine de cette étude.

3.1. Architecture matérielle de Magali

Le circuit Magali [8] est un système sur puce dédié au traitement de la couche physique des protocoles radios OFDMA, avec un intérêt particulier pour le 3GPP LTE-Advanced comme application de référence. Il est composé de différents blocs de calculs hétérogènes, avec des degrés de programmation variables, allant de blocs configurables (ex. le bloc fft avec la taille d'une FFT et le masque utilisé pour une modulation OFDM) jusqu'à des DSPs programmables en C. Les communications entre les blocs sont effectuées au travers d'un réseau sur puce maillé à deux dimensions. La configuration et le contrôle principal sont confiés à un CPU ARM.
Magali offre des mécanismes de contrôle distribué permettant l'enchaînement de séquences de programmes sur chacun des blocs matériels, limitant ainsi le nombre de reconfigurations effectuées par le CPU dans le cas d'applications complexes. Ils sont constitués d'un jeu d'instructions limité adapté à un ordonnancement statique, le dynamisme étant reporté au contrôleur central. Pour plus de détails sur ces mécanismes, se référer à l'article [9].

3.2. Modèles d'exécutions existants pour Magali

Le développement d'applications sur la plateforme Magali a donné lieu à l'expérimentation de plusieurs approches ayant différents modèles de programmation, de calcul et d'exécution. Nous allons nous concentrer sur les modèles d'exécution adressant le contrôleur centralisé et les contrôleurs distribués de chacune de ces approches.

Approche par phases

Tout d'abord, le modèle de programmation proposé lors de la création de la plateforme [8] suit une approche par phases. L'application est découpée selon plusieurs phases, chacune correspondant à un graphe flot de données statique. Durant une phase, les mécanismes de contrôle

distribués sont en charge de l'ordonnancement. La fin de chaque phase est synchronisée avec le contrôleur centralisé, qui est en charge de l'ordonnancement entre les phases. Des travaux suivant ce modèle proposent l'automatisation de cette tâche à l'aide d'un modèle abstrait de la plateforme et de l'application [19].

Approche machine virtuelle radio

L'utilisation d'une machine virtuelle radio [22] pour améliorer la portabilité et la flexibilité a été expérimentée avec la machine virtuelle LUA. Le modèle de calcul utilisé dans cette approche étant dynamique, il ne permet pas de réaliser un ordonnancement statique. L'ordonnancement de tous les blocs est donc pris en charge par le contrôleur centralisé, résultant en un grand nombre de synchronisations avec le CPU.

3.3. Proposition : modèle de contrôle par *threads*

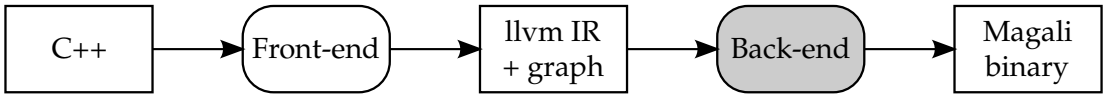


FIGURE 1 – Schéma d'ensemble du flot de compilation proposé. Cet article ce concentre sur le *back-end* générant le code du modèle d'exécution proposé.

Ce travail se situe dans le cadre du développement d'une nouvelle chaîne de compilation pour les plateformes hétérogènes visant la radio logicielle. Cette chaîne est illustrée sur la Fig. 1. Le C++ est utilisé comme langage source, enrichi d'une bibliothèque permettant de représenter un graphe flot de données paramétrique. Elle utilise l'infrastructure de compilation LLVM [1] ainsi que son format intermédiaire, enrichi d'une représentation du graphe flot de données. La compilation d'applications flot de données à l'aide de l'infrastructure LLVM se rapproche de ORCC [13]. Cette chaîne utilise le format intermédiaire LLVM comme format d'échange, ce format étant compilé à la volée lors de l'exécution sur la plateforme matérielle. Notre approche se distingue par l'utilisation d'une compilation statique afin de limiter le surcoût lors de l'exécution. Le *back-end* supportant la plateforme Magali est décrit dans l'article [10]. Ce *back-end* est en charge de la génération du code de contrôle suivant le modèle d'exécution proposé.
Cette chaîne a été développée afin de répondre à la problématique du développement visant les plateformes hétérogènes telles que Magali, avec des applications de télécommunications requiérant plus de flexibilité pour supporter les derniers protocoles radio ainsi que la radio cognitive. Ce besoin de flexibilité a mené à l'apparition de modèles de calcul paramétriques [4, 11, 12]. Ce travail décrit le modèle d'exécution mis au point pour supporter l'un de ces modèles de calcul : SPDF [12]. Afin de comprendre l'intérêt de développer ce nouveau modèle d'exécution, nous allons tout d'abord décrire le modèle de calcul adopté.
Un exemple de graphe SPDF est décrit dans la Fig. 2. Les valeurs sur chacun des arcs défini le nombre d'éléments produits ou consommés lors de l'exécution d'un acteur. En plus de valeurs entières, ce nombre d'éléments peut aussi être un paramètre symbolique dont la valeur est produite par un acteur (ex. l'acteur A produit le paramètre p).
L'intérêt de SPDF dans le cadre de notre compilateur est la possibilité de générer un *ordonnancement quasi-statique* à partir du graphe. L'ordonnancement de l'acteur B sur la Fig.2 est défini comme $(\texttt{pop p};(\texttt{B}^{\texttt{p}};\texttt{push q})^2)$, ce qui veut dire : récupérer la valeur du paramètre p, répéter

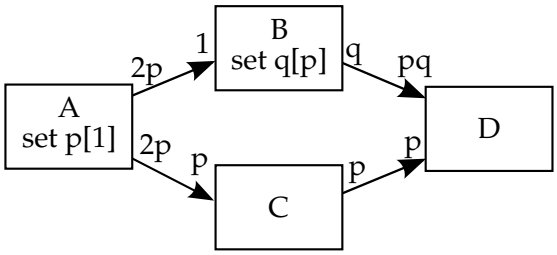


FIGURE 2 – Graphe flot de données paramétrique ordonnançable [12], p et q étant des *paramètres* variant lors de l'exécution.

deux fois le sous-ordonnancement suivant : exécuter p fois l'acteur B, produire la valeur du paramètre q.

approche par threads

Plusieurs modèles d'exécution associés à SPDF ont déjà été proposés pour d'autres plateformes. En premier lieu, Fradet et. al. [12] proposent avec leur modèle de calcul de générer un réseau de distribution des données sous la forme d'un graphe flot de données dynamique, supporté par un ordonnanceur dynamique. Un autre modèle d'exécution pour une plateforme *many-cores* propose l'utilisation d'un ordonnancement par *slots* [2], l'évaluation des prochains acteurs exécutés se faisant pendant l'exécution du *slot* actuel. Ces modèles conservent un ordonnancement dynamique et font l'hypothèse d'un contrôleur le supportant. Les contrôleurs distribués présents sur la plateforme Magali, essentiels pour atteindre le niveau de performance visé par la plateforme, ne supportent cependant qu'un ordonnancement statique. Il est donc nécessaire de développer un nouveau modèle d'exécution adapté.
Le modèle d'exécution permettant de supporter SPDF sur la plateforme Magali est basé sur un ordonnancement partagé entre le contrôleur central et les contrôleurs distribués. Le contrôleur central est en charge de la synchronisation des paramètres (ex. $(\mathtt{pop\ p}; (S_1; \mathtt{push\ q})^2)$), et les contrôleurs distribués supportent les parties statiques, c'est à dire lorsque les paramètres sont invariants (ex. $S_1 : (B^p)$). Pour suivre l'état d'avancement de chacun des ordonnancements, un fil d'exécution (*thread*) est créé sur le CPU pour chaque bloc matériel, et les primitives de synchronisation du système d'exploitation (ECOS) sont utilisées pour synchroniser les paramètres.

4. Résultats

Afin de comprendre l'influence des différentes approches sur les temps d'exécution, nous allons tout d'abord décrire les caractéristiques temporelles offertes par les différentes approches au travers de chronogrammes, puis présenter les applications de test ainsi que les résultats obtenus.

4.1. Analyse des chronogrammes

Dans cette analyse des performances, nous détaillons les différentes étapes d'une exécution au travers de chronogrammes du point de vue du contrôleur. Pour illustrer ces temps, nous utilisons l'application FFT illustré sur la Fig. 3.

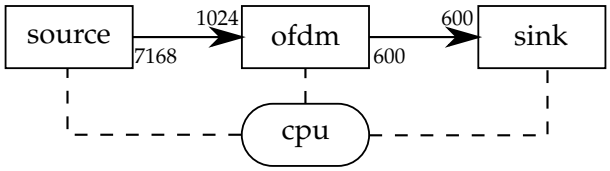


FIGURE 3 – Application FFT compilée pour la plateforme Magali.

Approche par phases

Le chronogramme de la Fig. 4 illustre le fonctionnement d'une application par phases. Les zones grisées représentent les périodes pendant lesquelles un bloc est activé, c'est à dire depuis le démarrage du bloc par le contrôleur central jusqu'à son arrêt. Les temps t_{ei} et t_{si} sont respectivement le temps d'arrêt et de démarrage du bloc i. Pour le contrôleur central (CPU), δ_{ci} est le délai de changement de contexte pour la prise en compte de l'interruption provenant du bloc i, et δ_{si} le délai de configuration et de démarrage du bloc i par le contrôleur principal.

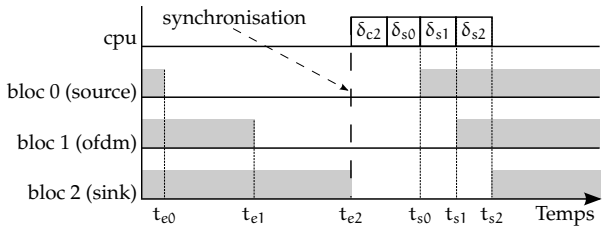


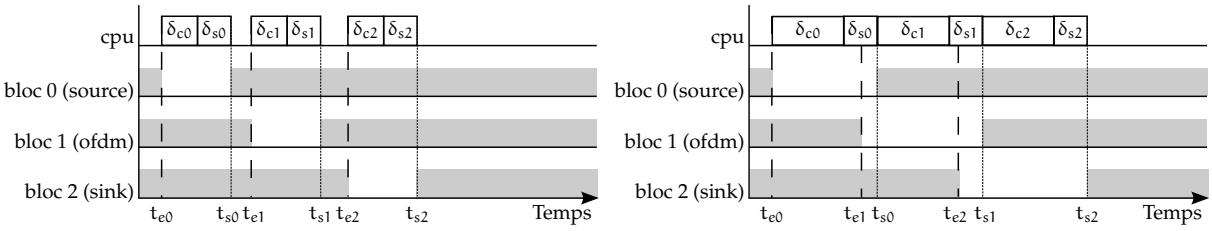
FIGURE 4 – Chronogramme d'une application contrôlée par phases.

On observe sur la Fig. 4 le point de synchronisation (trait pointillé à t_{e2}) introduit par le fonctionnement par phases. Le contrôleur central est inactif depuis le démarrage du dernier bloc jusqu'à ce point de synchronisation. Il existe donc une opportunité de reconfiguration en temps masqué. Cette reconfiguration peut se faire à partir de l'arrêt d'un bloc sur le temps de non recouvrement entre deux blocs consécutifs, c'est à dire le temps pendant lequel un bloc est actif alors que le bloc le précédent dans l'application est inactif. Nous désignerons ce temps par la suite comme $\delta_{e0} = t_{e1} - t_{e0}$. Nous allons chercher à minimiser le délai entre l'arrêt du premier bloc et le démarrage du dernier bloc. Dans le cas de l'approche par phase, ce délai est égal à (1).

$$t_{sn} - t_{e0} = \sum_{i=0}^{n-1} \delta_{ei} + \delta_{cn} + \sum_{i=0}^{n} \delta_{si} \tag{1}$$

Approche par threads

L'approche par *threads* vise à tirer parti de cette opportunité de calcul en temps masqué en séparant le contrôle de chaque bloc. Pour ceci, nous associons un fil d'exécution à chaque bloc sur le contrôleur principal. Ces fils d'exécution s'exécutent en concurrence sur un système d'exploitation ECOS, avec une synchronisation entre chaque bloc et son fil d'exécution réalisée à l'aide d'un sémaphore.
Le chronogramme de la Fig. 5a montre le fonctionnement de notre approche lorsque les délais de configuration et de démarrage sont inférieurs au délai de non recouvrement. Dans ce cas,



(a) $\delta_{ei} > \delta_{ci} + \delta_{si}$ (b) $\delta_{ei} < \delta_{ci} + \delta_{si}$

FIGURE 5 – Chronogramme d'une application contrôlée par *threads*.

nous obtenons bien un gain sur le temps d'exécution, en tirant parti du temps précédemment masqué par la synchronisation de l'approche par phase. Le délai à minimiser est ici égal à (2).

$$t_{sn} - t_{e0} = \sum_{i=0}^{n-1} \delta_i^e + \delta_{cn} + \delta_{sn} \tag{2}$$

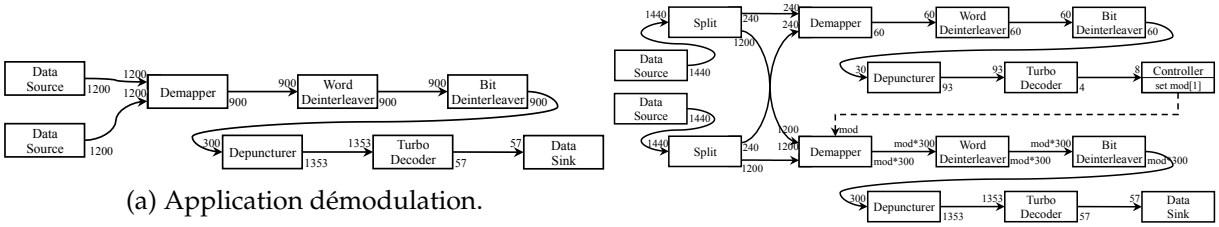
La Fig. 5b illustre quand à elle le cas où le délai de recouvrement est inférieur au délai de configuration et de démarrage. Dans ce cas l'approche peut être désavantageuse, le contrôleur central sérialisant le lancement des blocs avec un surcoût dû au temps de configuration pour chaque fil d'exécution. Le délai total est ici de (3).

$$t_{sn} - t_{e0} = \sum_{i=0}^{n} \delta_{ci} + \sum_{i=0}^{n} \delta_{si} \tag{3}$$

Les temps composant le calcul des délais (1),(2) et (3) sont dépendants du modèle d'exécution utilisé. On ne peut donc pas conclure à partir des valeurs théoriques quant à l'approche la plus appropriée. Nous avons donc confronté ces approches à l'expérimentation.

4.2. Expérimentations

Afin d'évaluer les différents modèles d'exécution sur des applications réalistes, nous avons extrait des parties représentatives du protocole LTE-Advanced pour illustrer l'influence du modèle d'exécution sur les performances temporelles, et les avons découpé en 3 applications. L'application FFT utilisée pour illustrer les chronogrammes de la section précédente est présentée sur la Fig. 3. L'application *démodulation* (Fig. 6a) permet d'évaluer l'influence du nombre de blocs sur les performances. L'application *démodulation paramétrique* (Fig. 6b) montre quand à elle l'influence de la gestion des paramètres sur chacune des approches.



(a) Application démodulation.

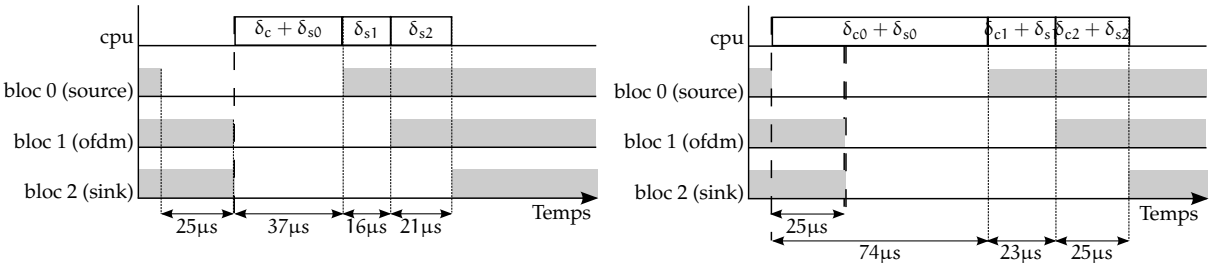
(b) Application démodulation paramétrique.

FIGURE 6 – Applications de test compilées pour la plateforme Magali.

L'ensemble des expérimentations sont réalisées sur un modèle de niveau transactionnel (*TLM*) de la plateforme Magali. Une machine virtuelle QEMU permet d'émuler le CPU ARM pour prendre en compte le temps d'exécution du contrôleur central.

4.3. Performances temporelles

Suite à l'analyse des chronogrammes, nous avons extrait un certain nombre de délais, afin de comprendre les temps d'exécution obtenus par la suite. Ces délais sont présentés dans les chronogrames des Fig. 7a et Fig. 7b pour les approches par phase et par *threads*.



(a) Contrôle par phases.

(b) Contrôle par *threads*.

FIGURE 7 – Chronogramme de l'application FFT.

Nous sommes ici dans le cas où le temps de recouvrement est inférieur au temps configuration et de démarrage. De plus, les interruptions des blocs 2 et 3 sur le contrôleur centralisé ralentissent le changement de contexte lors de la configuration du premier bloc. Au final, on observe un surcoût à l'utilisation de cette méthode, qui se reflète dans les résultats de performances présentés par la suite.

Les temps d'exécutions des différentes approches sur chacune des trois applications sont présentés dans la Tab. 1. Les approches par phase, par machine virtuelle et par *threads* correspondent aux approches décrites dans les sections 3.2 et 3.3. La version manuelle est, elle, créée sur la base du code généré par le compilateur en minimisant le temps total de reconfiguration.

Application	phase	machine virtuelle	*threads*	manuelle
FFT	149 µs	500 µs (+ 236%)	168 µs (+ 13%)	149 µs (- 0%)
démodulation	180 µs	-	283 µs (+ 57%)	180 µs (- 0%)
démod. paramétrique	419 µs	-	558 µs (+ 33%)	288 µs (- 31%)

TABLE 1 – Performances obtenues selon les différents approches.

L'approche par phase sert de point de référence pour l'évaluation des performances. On peut tout d'abord observer que l'approche par machine virtuelle possède un surcoût très important, ceci étant dû à l'absence de support des contrôleurs distribués, qui résulte en un nombre important de synchronisations coûteuses avec le contrôleur principal.

On remarque que l'approche par *threads* génère un surcoût proportionnel au nombre de blocs utilisés pour les deux premières applications. Ceci est dû à un temps de recouvrement inférieur au temps de configuration des blocs, qui vient valider l'approche initiale par phase. Dans le

cas de la troisième application, ce surcoût est équilibré par l'utilisation d'un seul graphe pour l'ensemble de l'application, contrairement à l'application par phases découpée entre la partie statique et la partie paramétrique du graphe. Les blocs sources n'étant pas influencés par le paramètre, notre approche permet de limiter le nombre de blocs à reconfigurer.
La version manuelle est, elle, équivalente à l'approche par phase pour les deux premières applications. En effet, les temps de recouvrement étant faibles, on utilise un seul fil d'exécution. Dans le cas de la troisième application, le gain est obtenu grâce à l'utilisation du graphe paramétrique limitant le nombre de blocs à reconfigurer lors de l'exécution.
Au final, ces résultats illustrent l'importance d'adapter le modèle d'exécution, dans le cas particulier des plateformes à contrôle centralisé, non seulement à la plateforme matérielle, mais aussi aux caractéristiques de l'application. Ici, le choix de la granularité optimale du contrôle est dépendant des caractéristiques de l'application. Dans le cas où le temps de recouvrement est supérieur au temps de changement de contexte et de configuration (Fig. 5a), l'approche par thread est la plus efficace. Dans le cas contraire, une approche itérative doit être utilisée pour trouver le meilleur compromis, un changement du temps de démarrage d'un bloc pouvant influencer le temps de non recouvrement.

5. Conclusion

Les tendances actuelles en terme d'architecture matérielle, particulièrement dans le domaine de l'embarqué, vont vers des architectures hétérogènes et distribuées. Ces architectures posent le problème du contrôle distribué, associé à des contraintes de réactivité fortes. Dans cette étude, nous nous sommes concentrés sur les applications traitement du signal bénéficiant d'un modèle de calcul flot de données. Ce modèle permet d'expliciter les dépendances entre les traitements, permettant ainsi d'extraire des informations pour guider le modèle d'exécution.
Nous avons détaillé le fonctionnement de plusieurs modèles d'exécution pour la plateforme Magali, et introduit un nouveau modèle ajoutant le support des paramètres à cette plateforme. Notre étude s'est concentré l'influence du temps de recouvrement sur les performances du modèle d'exécution. Nous avons ensuite observé sur un cas pratique, le protocole LTE-Advanced, les performances de ces différents modèles. Ces expériences ont permis de montrer l'importance d'une part d'avoir un modèle d'exécution adapté à la plateforme, mais aussi de tenir compte des caractéristiques temporelles de l'application dans le choix de ce modèle.
Les travaux futurs devront se concentrer d'une part sur la possibilité d'automatiser le choix de ce modèle, pour se rapprocher des performances optimales obtenues manuellement. Un autre axe d'étude potentiellement intéressant serait le développement d'un système d'exploitation spécialisé aux applications flots de données, ce qui permettrait de réduire les temps de reconfiguration et ainsi obtenir de nouvelles opportunités de travail en temps masqué.

Bibliographie

1. http ://llvm.org : The llvm compiler infrastructure.
2. Bebelis (V.), Fradet (P.), Girault (A.) et Lavigueur (B.). – A Framework to Schedule Parametric Dataflow Applications on Many-Core Platforms. *In : 17th workshop on Compilers for Parallel Computing, CPC 2013*. – Lyon, FR, juillet 2013.
3. Berg (H.), Brunelli (C.) et Lucking (U.). – Analyzing models of computation for software defined radio applications. *In : International Symposium on System-on-Chip*, pp. 1–4. – Tampere, Finland, novembre 2008.

4. Bhattacharya (B.) et Bhattacharyya (S.). – Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, vol. 49, n10, 2001, pp. 2408–2421.
5. Bougard (B.), De Sutter (B.), Verkest (D.), Van der Perre (L.) et Lauwereins (R.). – A Coarse-Grained Array Accelerator for Software-Defined Radio Baseband Processing. *IEEE Micro*, vol. 28, n4, juillet 2008, pp. 41–50.
6. Castrillon (J.), Schürmans (S.), Stulova (A.), Sheng (W.), Kempf (T.), Leupers (R.), Ascheid (G.) et Meyr (H.). – Component-based waveform development : the Nucleus tool flow for efficient and portable software defined radio. *Analog Integrated Circuits and Signal Processing*, vol. 69, n2-3, juin 2011, pp. 173–190.
7. Clermidy (F.), Bernard (C.), Lemaire (R.), Martin (J.), Miro-Panades (I.), Thonnart (Y.), Vivet (P.) et Wehn (N.). – A 477mW NoC-based digital baseband for MIMO 4G SDR. *In : IEEE International Solid-State Circuits Conference - (ISSCC)*, pp. 278–279. – San Francisco, CA, février 2010.
8. Clermidy (F.), Lemaire (R.), Popon (X.), Ktenas (D.) et Thonnart (Y.). – An Open and Reconfigurable Platform for 4G Telecommunication : Concepts and Application. *In : 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, pp. 449–456. – Patras, Greece, août 2009.
9. Clermidy (F.), Lemaire (R.) et Thonnart (Y.). – A Communication and configuration controller for NoC based reconfigurable data flow architecture. *In : 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pp. 153–162. – San Diego, CA, mai 2009.
10. Dardaillon (M.), Marquet (K.), Risset (T.), Martin (J.) et Charles (H.-p.). – Compilation for heterogeneous SoCs : bridging the gap between software and target-specific mechanisms . *In : workshop on High Performance Energy Efficient Embedded Systems - HIPEAC*. – Vienna, Austria, janvier 2014.
11. Desnos (K.), Pelcat (M.), Nezan (J.-F.), Bhattacharyya (S. S.) et Aridhi (S.). – PiMM : Parameterized and Interfaced dataflow Meta-Model for MPSoCs runtime reconfiguration. *In : International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS)*. pp. 41–48. – Samos, Greece, juillet 2013.
12. Fradet (P.), Girault (A.) et Poplavko (P.). – SPDF : A schedulable parametric data-flow MoC. *In : Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 769–774. – Dresden, Germany, mars 2012.
13. Gorin (J.), Wipliez (M.), Preteux (F.) et Raulet (M.). – A portable Video Tool Library for MPEG Reconfigurable Video Coding using LLVM representation. *In : Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pp. 183–190. – Edinburgh, Scotland, octobre 2010.
14. Goubier (T.), Sirdey (R.), Louise (S.) et David (V.). – ΣC A Programming Model and Language for Embedded Manycores. *In : Algorithms and Architectures for Parallel Processing - 11th International Conference, ICA3PP*, pp. 385–394. – Melbourne, Australia, octobre 2011.
15. Horrein (P.-H.), Hennebert (C.) et Pétrot (F.). – Integration of GPU Computing in a Software Radio Environment. *Journal of Signal Processing Systems*, vol. 69, n1, décembre 2011, pp. 55–65.
16. Jalier (C.), Lattard (D.), Jerraya (A.), Sassatelli (G.), Benoit (P.) et Torres (L.). – Heterogeneous vs homogeneous MPSoC approaches for a mobile LTE modem. *In : Conference on Design, Automation and Test in Europe*, pp. 184–189. – Dresden, Germany, mars 2010.
17. Limberg (T.), Winter (M.), Bimberg (M.), Klemm (R.), Matus (E.), Tavares (M. B.), Fettweis (G.), Ahlendorf (H.) et Robelly (P.). – A fully programmable 40 GOPS SDR single chip baseband for LTE/WiMAX terminals. *In : 34th European Solid-State Circuits Conference - ESSCIRC*. pp. 466–469. – Edinburgh, Scotland, septembre 2008.

18. Lin (Y.), Mullenix (R.), Woh (M.), Mahlke (S.), Mudge (T.), Reid (A.) et Flautner (K.). – SPEX : A programming language for software defined radio. *In : SDR Forum Technical Conference*, pp. 13 – 17. – Orlando, Florida, novembre 2006.
19. Mrabti (A. E.), Sheibanyrad (H.), Rousseau (F.), Petrot (F.), Lemaire (R.) et Martin (J.). – Abstract Description of System Application and Hardware Architecture for Hardware/-Software Code Generation. *In : 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, pp. 567–574. – Patras, Greece, août 2009.
20. Palkovic (M.), Raghavan (P.), Li (M.), Dejonghe (A.), Van der Perre (L.) et Catthoor (F.). – Future Software-Defined Radio Platforms and Mapping Flows. *IEEE Signal Processing Magazine*, vol. 27, n2, mars 2010, pp. 22–33.
21. Ramacher (U.). – Software-Defined Radio Prospects for Multistandard Mobile Phones. *Computer*, vol. 40, n10, 2007, pp. 62–69.
22. Risset (T.), Ben Abdallah (R.), Fraboulet (A.) et Martin (J.). – *Digital Front-End in Wireless Communications and Broadcasting*, chap. Programming models and implementation platforms for software defined radio configuration, pp. 650–670. – Cambridge University Press, 2011.
23. Schulte (M. J.), Glossner (J.), Mamidi (S.), Moudgill (M.) et Vassiliadis (S.). – A low-power multithreaded processor for baseband communication systems. *Computer Systems : Architectures, Modeling, and Simulation*, vol. 3133, nLNCS, 2004, pp. 333–346.
24. Wang (P. H.), Collins (J. D.), Chinya (G. N.), Jiang (H.), Tian (X.), Girkar (M.), Yang (N. Y.), Lueh (G.-y.) et Wang (H.). – EXOCHI : architecture and programming environment for a heterogeneous multi-core multithreaded system. *In : ACM SIGPLAN conference on Programming language design and implementation - PLDI*, p. 156. – San Diego, CA, juin 2007.
25. Woh (M.), Harel (Y.), Mahlke (S.), Mudge (T.), Chakrabarti (C.) et Flautner (K.). – SODA : A Low-power Architecture For Software Radio. *In : 33rd International Symposium on Computer Architecture, ISCA*. pp. 89–101. – Boston, MA, juin 2006.
26. Woh (M.), Lin (Y.), Seo (S.), Mahlke (S.), Mudge (T.), Chakrabarti (C.), Bruce (R.), Kershaw (D.), Reid (A.), Wilder (M.) et Flautner (K.). – From SODA to scotch : The evolution of a wireless baseband processor. *In : 41st IEEE/ACM International Symposium on Microarchitecture*. pp. 152–163. – Como, Italy, novembre 2008.